Comment l'utiliser :

- Les fonctions utilisables : sin(), cos(), tan(), sqrt(), exp(), log(), abs(), sinc(), entier(),
- Pour les valeurs négatives toujours utiliser les parenthèses ex : sin(-2) => sin((-2)) ou -4 => (-4)
- Pour les puissances utiliser ** ex : x puissance y => x**y
- Pour les nombre decimaux utiliser des '.' et non des ','
- toutes les lettres majuscules sont considérées comme des minuscules

1. Accueil :

L'écran d'accueil facilite l'accès aux fonctionnalités comme le mode d'emploi, les remerciements ou le lancement du mode d'affichage, avec une version 3D en développement.

2. Retour:

Pour revenir aux menus précédents sans relancer le programme, soit un clique sur l'icône de maison, soit en utiliser la touche "Backspace" ou "Retour".

3. Bande haute :

Permet d'afficher et de modifier jusqu'à 20 expressions avec des fonctionnalités interactives.

- Changement de couleur : Sélectionnez une nouvelle couleur en temps réel via l'icône de roue.
- Cacher une fonction : Masquez ou affichez une fonction selon vos besoins en temps réel.
- Supprimer une fonction : Cliquez sur l'icône de poubelle pour supprimer une expression.
- Modifier les bornes : Ajustez les bornes d’une expression via une fenêtre de texte pour plus de flexibilité.

4. Affichage des fonctions :

L'évaluation des fonctions met à jour dynamiquement leur représentation graphique.

- Zoom : Utilisez la molette de la souris pour ajuster l'affichage du graphique.
- Centrage : Réinitialisez l'affichage avec la touche "C" ou le bouton "Recentrer" à venir.
- Translation du repère : Glissez sur le graphique avec un clic droit pour parcourir la courbe.
- Évaluateur en x : Activez avec un clic gauche pour afficher les coordonnées d'un point sur la courbe.

Nos structures :

Pour développer notre grapheur nous avons utilisé plusieurs structures notamment pour coder les fonctions sous forme d'arbre mais également pour retourner les différentes erreurs possibles. Cette partie nous a été fournie au début du projet, nous y avons fait quelques modifications que nous allons vous présenter. Le fichier qui nous a été fourni contenait 6 structures différentes :

- arbre : prend en paramètre un jeton (Typejeton), un arbre suivant et un arbre précédent.
- typejeton : prend en paramètre le type de valeur du jeton (typelexem) et la valeur du jeton (typevaleur).
- typevaleur : prend en paramètre si la valeur du jeton, qui peut être un réel, une fonction, un opérateur et nous avons décidé d'ajouter également la possibilité d'inclure une variable (char) qui prendra comme valeur x ou y afin de pouvoir effectuer le graphe d'une fonction à deux variables.
- typefonction : prend en paramètre une fonction mathématique (exp, sin, cos,...)
- typeoperateur : prend en paramètre un opérateur (plus, moins ; fois, div, puissance)
- typelexem : prend en paramètre le nom du le type de valeur du jeton.

Nous avons ajouté également une structure code_erreur qui associe un nom d'erreur à un code d'erreur, par exemple : RACINE_NEGATIVE=301, il est impossible de calculer la racine d'un nombre négatif donc si l'utilisateur demande en entrée d'en faire une le code renverra cette erreur.

Explication de la fonction Analyse_Lexicale

La fonction 'Analyse_Lexicale' transforme une expression mathématique sous forme de chaîne de caractères en un tableau de jetons (tokens). Elle permet de décomposer l'expression en éléments que l'on peut facilement manipuler pour des opérations d'analyse syntaxique ultérieures.

Ce processus de tokenisation est une étape préalable fondamentale pour la future analyse et évaluation de l'expression mathématiques.

Fonctionnement global

La fonction prend quatre paramètres:

- 'TabToken' : tableau qui contiendra les tokens générés
- 'Expression' : la chaîne de caractères représentant l'expression mathématique
- 'erreur' : pointeur vers une variable qui contiendra le code d'erreur en cas de problème
- 'Dimension' : indique si l'expression est à 1 ou 2 variables (0 pour une dimension, 1 pour deux dimensions)

Important: L'analyseur lexical considère les majuscules comme des minuscules.

Important: Pour les nombres décimaux, utiliser des points et non pas des virgules.

Sous-fonctions utilisées

La fonction 'Analyse_Lexicale' suit une séquence de traitement en trois étapes principales, chacune utilisant une sous-fonction dédiée:

1. 'ExpressionSansLesEspaces' Cette fonction supprime tous les espaces de l'expression mathématique.

2. 'MultiplicationImplicite' Cette fonction détecte et ajoute les opérateurs de multiplication implicites. Par exemple, elle transforme '2x' en '2*x'.

3. 'DecompositionToken' Cette fonction décompose l'expression en tokens individuels et les stocke dans le tableau 'TabToken', en tenant compte du paramètre de dimension.

Processus détaillé avec exemples

Prenons l'expression '"3 + cos(5x)"' comme exemple.

Étape 1: Suppression des espaces

'ExpressionSansLesEspaces' transforme '"3 + cos(5x)"' en '"3+cos(5x)"'.

Étape 2: Ajout des multiplications implicites

'MultiplicationImplicite' détecte les multiplications implicites entre chiffres et variables. Elle transforme '"3+cos(5x)"' en '"3+cos(5*x)"'.

Étape 3: Création des tokens

'DecompositionToken' décompose l'expression en tokens:

- '3' → Token de type REEL avec valeur 3.0
- '+' → Token de type OPERATEUR avec valeur PLUS
- 'cos' → Token de type FONCTION avec valeur COS
- '(' → Token de type PAR_OUV
- '5' → Token de type REEL avec valeur 5.0
- '*' → Token de type OPERATEUR avec valeur FOIS
- 'x' → Token de type VARIABLE avec valeur 'x'
- ')' → Token de type PAR_FERM
- Ajout d'un token FIN à la fin

Comment 'DecompositionToken' identifie les tokens

La fonction 'DecompositionToken' analyse caractère par caractère et utilise plusieurs sous-fonctions pour identifier correctement chaque token:

- 'TokenReelPositif': Crée un token pour les nombres positifs
- 'TokenReelNegatif': Gère les nombres négatifs (ex: "(-2.5)")
- 'TokenOperateur': Identifie les opérateurs (+, -, *, /, **)
- 'TokenFonction': Identifie les fonctions (sin, cos, abs, etc.)
- 'TokenVariable': Identifie les variables (x, y) en fonction du paramètre 'Dimension'

Gestion des variables selon la dimension

La fonction 'TokenVariable' utilise le paramètre 'Dimension' pour savoir quelles variables sont autorisées:

- Si 'Dimension' = 0: seule la variable 'x' est autorisée
- Si 'Dimension' = 1: les variables 'x' et 'y' sont autorisées

Gestion des erreurs

La fonction signale plusieurs types d'erreurs possibles:

- FONCTION_INCONNUE (101): Fonction non reconnue
- NOMBRE_INVALIDE (102): Format de nombre incorrect (ex: "1.2.3")
- CARACTERE_INCONNUE (103): Caractère non reconnu
- VARIABLE_INCONNUE (104): Variable non autorisée selon la dimension

Exemple complet

Pour l'expression '"2x + sin(3.5)"' avec 'Dimension' = 0:

1. Suppression des espaces: '"2x+sin(3.5)"'
2. Ajout des multiplications implicites: '"2*x+sin(3.5)"'
3. Création des tokens:
    - REEL (2.0)
    - OPERATEUR (FOIS)
    - VARIABLE ('x')
    - OPERATEUR (PLUS)
    - FONCTION (SIN)
    - PAR_OUV
    - REEL (3.5)
    - PAR_FERM
    - FIN

Analyse syntaxique

createEmptyNode

utilité: Renvoie un arbre vide

Prend en paramètre:

-Rien

Fonctionnement de la fonction:

-Créée de fg et fd vides

buildExpressionTree

utilité: Construit récursivement l'arbre à partir d'un tableau de jetons

Prend en paramètre:

-un tableau de jetons -l'indice de la première case à analyser -l'indice de la dernière case à analyser -l'erreur récupérée par le dernier appel récursif (l'erreur par défaut est 0).

Fonctionnement de la fonction:

Si présence d'erreur au dernier appel récursif: renvoie un arbre vide au bout de la branche

Si l'indice du début > indice de fin : renvoie erreur = MEMBRE_VIDE

S'il y a un opérateur: scission de l'arbre sur l'opérateur en fg et fd.

Sinon, renvoie une erreur dans les cas suivant

- PROBLEME_PARENTHESES_FONCTIONS : manque une parenthèse après une fonction ou si la parenthèse ne se ferme pas
- MEMBRE_VIDE : si un opérateur n'est pas entouré d'objets traitables, si des parenthèses sont vides
- PARENTHESE_FERMEE_1_ER_JETON : si une parenthèse est fermée sans avoir été ouverte
- PROBLEMES_NOMBRE_PARENTHESES : si le nombre de parenthèses ouvrante est différent du nombre de parenthèses fermentes
- PROBLEME_APRES_REEL_OU_VARIABLE : Supposé impossible
- ABSENCE_FIN : si le jeton "FIN" est manquant

findLowestPriorityOperator

Utilité: Renvoie l'opérateur où faire la scission entre le fg et le fd.

Prend en paramètre:

-un tableau de jetons -l'indice de la première case à analyser -l'indice de la dernière case à analyser -l'erreur récupérée par le dernier appel récursif (l'erreur par défaut est 0).

Fonctionnement de la fonction:

Initialise la profondeur de complexité des parenthèses à Met par défaut l'indice de scission à -1 .

indiceOperateurMinimal = -1 si pas d'opérateur dans la foncti prioritéOperateurMinimal représente l'opérateur sur lequel séparer le tableau de jetons.

Parcours le tableau case par case et regarde s'il y a un opérateur

Si la case est un opérateur, que la profondeur actuelle des parenthèses est nulle et que l'opérateur de la case actuelle est plus propice à la scission: indiceOperateurMinimal = indice actuel prioriteOperateurMinimal = case actuelle

Si la case est une PAR_OUV: la profondeur liée aux parenthèses augmente

Si la case est une PAR_FERM: la profondeur liée aux parenthèses diminue

Si la profondeur des parenthèses n'est pas nulle (=> nombre de PAR_OUV != nombre de PAR_FERM erreur = PROBLEMES_NOMBRE_PARENTHESES

findExpressionLength

Utilité: Renvoie la taille du tableau

Prend en paramètre:

-Un tableau de jetons

Fonctionnement de la fonction:

Parcours le tableau et si la case est une case fin, la fonction s'arrête Si aucune FIN n'est trouvée alors fin = -1

checkParenthesesBalance

Utilité: Renvoie un booléen par rapport à l'équilibre du nombre de parenthèses.

Prend en paramètre:

-indice du début du tableau à analyser -indice de fin du tableau à analyser -le tableau de jetons

Fonctionnement de la fonction:

Parcours le tableau et compte le nombre de PAR_OUV et de PAR_FERM

buildSyntaxTree

Utilité: Construit l'arbre syntaxique complet à partir d'une séquence de jetons

Prend en paramètre:

-le tableau de jetons -la dernière erreur renvoyée par buildExpressionTree (0 pour le premier appel)

Fonctionnement de la fonction:

Vérifie si le jeton FIN existe S'il existe alors l'abre se construit en appelant buildExpressionTree Sinon erreur = ABSENCE_FIN

PARTIE EVALUATEUR :

L'objectif de l'évaluateur est de calculer une fonction (qui est sous la forme d'un arbre) à partir de variable x et y données. Son rôle est essentiel pour tracer des courbes, car il permet d'obtenir les points à afficher sur le graphique. Notre code se divise en deux fonctions principales : evaluateur, notre fonction principale qui interprète l'arbre, et calculer_fonction, une fonction auxiliaire qui applique des fonctions mathématiques standards. Fonction evaluateur :

Elle a pour objectif de décoder l'arbre d'entrée. Elle prend en entréese :

- La fonction sous forme d'un arbre
- Les valeurs de x et y
- Le pointeur code_erreur. Elle retourne le résultat de la fonction d'entrée, calculé à partir des variables x et y. Cette fonction est récursive et possède quatre cas d'arrêt. Le premier est si le pointeur code_erreur change de valeur, cela signifie qu'une erreur est présente dans la fonction à calculer et qu'il faut donc stopper le décodage de l'arbre une fois l'apparition d'une erreur. Le deuxième cas d'arrêt est le si le jeton principal de l'arbre ne possède pas de type. Pour finir la fonction se stoppera si le jeton principal de l'arbre est un réel ou une variable, cela signifie qu'on est arrivé à une feuille de l'arbre. Une fois cela fait on regarde à l'aide d'une fonction switch le type du jeton principal de l'arbre d'entrée, s'il s'agit :
- D'une variable : on retourne la valeur de la variable qu'on a récupéré en entrée de fonction.
- D'un réel : on retourne la valeur du réel.
- Un opérateur : on utilise une méthode diviser pour régner pour calculer l'opération.
- D'une fonction : on utilise alors la deuxième fonction de notre programme. Fonction calculer_fonction :

Cette fonction prend en entrée le type de la fonction mathématique f, la variable x sur laquelle applique f et le pointeur code_erreur. Elle retourne l'application f(x). Les type des fonctions possibles sont les fonctions mathématiques universelles telles que sin, cos ou ln. On regarde d'abord à l'aide d'une fonction switch de quelle fonction il s'agit, puis on vérifie qu'il n'y ait pas d'erreur, avant de calculer f(x). En cas d'erreur : En cas d'erreur (ex: division par zéro, logarithme d'un nombre négatif...), la fonction retourne une valeur incorrecte et modifie le pointeur code_erreur pour signaler le problème. Cela permet par exemple d'arrêter l'évaluation ou d'afficher un message approprié à l'utilisateur.

Affichage graphique

1. Accueil

L'écran d'accueil permet de faciliter l'interaction avec l'utilisateur. Il donne accès aux différentes fonctionnalités, telles que la consultation du mode d'emploi, l'affichage des remerciements ou le lancement d'un mode d'affichage. À l'heure actuelle, le programme propose un affichage 2D, mais une version 3D pour les fonctions à deux variables est en cours de développement par nos ingénieurs.

2. Fonctionnalités de retour

Une fonctionnalité de retour permet à l'utilisateur de revenir aux menus précédents sans relancer le programme. Il peut revenir à l'écran d'accueil en cliquant sur l'icône en forme de maison dans les menus textes ou en appuyant sur la touche "Backspace" ou "Retour" du clavier.

3. Bande haute

L'utilisateur peut afficher jusqu'à 20 expressions. Par défaut, une fonction de base est présentée pour chaque cas. L'utilisateur peut ensuite cliquer sur une expression pour la modifier. De plus, lorsqu'il rédige dans l'une des trois zones de texte de la bande haute, il peut interagir avec les fonctions suivantes :

3.1 Changement de couleur

Le programme offre la possibilité de personnaliser la couleur des fonctions via un menu de sélection de couleur. Ce menu s'active en cliquant sur l'icône de roue de couleur. Une fois ouvert, il permet de changer la couleur des fonctions en temps réel (même en maintenant le clic gauche).

3.2 Cacher une fonction

Cette fonctionnalité permet à l'utilisateur de masquer ou d'afficher les fonctions de son choix en temps réel, en fonction de ses besoins.

3.3 Supprimer une fonction

L'utilisateur peut supprimer des expressions en cliquant sur l'icône de poubelle avec le clic gauche de la souris.

3.4 Modifier les bornes

Une fenêtre de texte permet à l'utilisateur de modifier les bornes d'une expression en temps réel, pour un gain de temps et plus de flexibilité.

4. Traitement de l'expression

Une fois l'expression saisie, elle est soumise à une analyse lexicale, puis syntaxique, afin de préparer son traitement ultérieur.

5. Affichage des fonctions

L'arbre généré lors de l'analyse est utilisé pour associer à chaque valeur de l'intervalle son image correspondante sur l'écran, grâce à la fonction d'évaluation. Ce processus se répète en temps réel, permettant ainsi des mises à jour dynamiques de l'affichage.

5.1 Zoom

Pour une meilleure analyse graphique, le programme permet de zoomer ou de dézoomer à l'aide de la molette de la souris. Cela modifie l'intervalle affiché de la fonction.

5.2 Centrage

La fonctionnalité de centrage permet de réinitialiser rapidement l'affichage. Il suffit d'appuyer sur la touche "C" du clavier. Un bouton "Recentrer" sera bientôt ajouté sur la bande droite de la fenêtre.

5.3 Translation du repère

En appuyant sur le bouton "clic droit" de la souris, l'utilisateur active un mode de "glissement" qui lui permet de parcourir le graphique de manière intuitive.

5.4 Évaluateur en x

L'évaluateur, activé par un clic gauche, permet à l'utilisateur de visualiser deux droites qui coupent la première courbe de la liste des expressions. Ces droites suivent la position de la souris sur l'axe des abscisses. Un affichage fournit la valeur de 'x' ainsi que la valeur de la fonction au point de clic.